Calcul de solutions équilibrées Pareto-optimales : application au problème de gestion de dépendances logicielles

Daniel Le Berre, Emmanuel Lonca, Pierre Marquis, Anne Parrain

Calcul de solutions équilibrées Pareto-optimales : application au problème de gestion de dépendances logicielles

Daniel Le Berre, Emmanuel Lonca, Pierre Marquis, Anne Parrain

Université Lille Nord de France, F-59000 Lille, France
Université d'Artois, CRIL, F-62300 Lens, France
CNRS, UMR8188, F-62300 Lens, France
{nom}@cril.fr

Résumé

Notre travail se situe dans le cadre de la gestion de dépendances logicielles. Nous présentons dans cet article des améliorations de l'approche décrite dans [2] concernant le calcul de solutions équilibrées dans ce contexte. Plus précisément, nous nous intéressons ici au calcul de solutions équilibrées Pareto-optimales. Nous comparons plusieurs codages de ce problème à base de variables booléennes, dont la résolution est confiée au prouveur Sat4j, et un codage utilisant une variable entière, dont la résolution est confiée au prouveur CPLEX.

Abstract

Our work deals with software dependency management problems. In this article, we present some improvements of the approaches presented in [2] for computing balanced solutions in this context. More precisely, our aim is to to compute Pareto optimal balanced solutions. We compare some encodings using Boolean variables, implemented on top of the Sat4j solver, and an encoding using integer variables, implemented on top of the CPLEX solver.

1 Introduction

Le problème de gestion de dépendances logicielles concerne l'installation de programmes informatiques modulaires. Un module peut nécessiter la présence d'autres modules pour pouvoir fonctionner, il peut entrer en conflit avec certains autres modules, et parfois il peut recommander l'installation de modules spécifiques pour pouvoir être utilisé au meilleur de ses capacités. Il s'agit de déterminer, lorsqu'on veut installer ou mettre à jour un programme informatique, les modules à installer et les modules à enlever. Le problème de décision associé est NP-complet [10, 18]. Plusieurs approches pour résoudre ce problème ont été proposées ces dernières années dans le cadre des dépendances GNU/Linux [6].

Le problème de gestion des dépendances se modélise de manière naturelle à l'aide de variables booléennes, en associant à chaque paquetage une variable. La valeur de vérité de la variable correspond alors à l'état du paquetage dans le système : installé ou non. Les dépendances entre paquetages peuvent être représentées par des clauses. On obtient donc une instance du problème SAT, qui peut être résolue par l'un des nombreux prouveurs disponibles. Le détail du codage en logique propositionnelle que nous utilisons est décrit dans [1].

Les problèmes de gestion de dépendances sont généralement sous-contraints. Le problème n'est pas de trouver une solution, mais une solution qui a un sens pour la personne qui gère le système, par exemple, qui conduit à installer uniquement les paquetages nécessaires à l'ajout d'une fonctionnalité au système, ou bien qui limite les changements aux seules versions des logiciels installés. Un ensemble de six critères a été défini par le projet européen Mancoosi [4] pour évaluer les solutions obtenues par les outils de gestion de dépendances. Le problème de gestion de dépendances logicielles se réduit alors à un problème d'optimisation, en variables booléennes, multi-critères, sous contraintes. Il existe diverses approches pour traiter ces problèmes en variables booléennes (formalismes pseudo-booléen [1], MaxSat [1], formalisme ASP [5], programmation linéaire [13], ...)

Dans le cadre du projet Mancoosi, les critères étaient considérés de manière lexicographique. Cela implique un ordre total entre les critères, ce qui ne correspond pas

toujours aux préférences de l'utilisateur, car l'optimisation d'un critère peut dégrader fortement la valeur prise à l'optimum pour un autre critère moins préféré. Nous nous sommes donc intéressés au calcul de solutions « équilibrées » [2], c'est-à-dire aussi proches que possible de la valeur optimale pour chacun des critères. Nous avons utilisé pour cela la norme de Tchebycheff [17, 19]. Nous avons proposé des algorithmes de calcul de solution équilibrée applicables pour des problèmes codés sous forme de contraintes dans le cas multi-critère. Dans cet article, nous comparons dans un premier temps les codages existants avec un nouveau codage permettant de linéariser la fonction objectif multi-critère considérée. Dans un second temps, nous proposons une solution pour ne conserver que des solutions Pareto-optimales parmi les solutions équilibrées, propriété qui n'est pas assurée pour les solutions optimales selon la norme de Tchebycheff.

2 Norme de Tchebycheff

La norme de Tchebycheff permet de caractériser des solutions (ou alternatives réalisables) équilibrées. Elle se base sur la notion de **point idéal**, qui est le vecteur des coûts optimaux pour chaque critère, et qui ne représente bien souvent pas une alternative admissible.

Définition 1 (point idéal)**.** *Soit S l'ensemble des solutions, et soit $c_1, c_2, ..., c_n$ les fonctions de coût associées aux n critères. Le point idéal $(c_1^*, c_2^*, ..., c_n^*) \in \mathbb{N}^n$ est le vecteur tel que $\forall i \in \{1, 2, ..., n\}$, $c_i^* = min(\{c_i(s) \mid s \in S\})$.*

Pour un critère donné, la différence entre la valeur pour une solution et celle du point idéal exprime le **regret** de choisir cette solution.

Définition 2 (regret)**.** *Soit $x^* = (c_1^*, c_2^*, ..., c_n^*)$ le point idéal d'un problème à n critères et $c_1, c_2, ..., c_n$ les fonctions de coût associées à ces critères. Soit $s \in S$ une solution du problème. Le regret de s pour le $i^{ème}$ critère est la valeur $r_i(s) = c_i(s) - c_i^*$*

Nous pouvons maintenant définir la méthode **min-max regret**, qui préfère parmi plusieurs alternatives celles qui offrent le plus petit regret maximal sur les critères qui composent leur vecteur.

Définition 3 (min-max regret)**.** *Soient s et s' deux alternatives admissibles, et $r_1, r_2, ..., r_n$ les fonctions exprimant le regret sur les n critères. La solution s est préférée à la solution s' par la méthode du **min-max regret** si et seulement si :*

$$max(\{r_i(s) \mid i \in 1, ..., n\}) < max(\{r_i(s') \mid i \in 1, ..., n\}).$$

Une variante de cette méthode où les regrets associés à chaque critère sont pondérés est nommée **méthode de Tchebycheff**.

Définition 4 (norme et méthode de Tchebycheff)**.** *Soient s et s' deux alternatives admissibles, et $r_1, r_2, ..., r_n$ les fonctions exprimant le regret sur les n critères. Soit $W = (w_1, w_2, ..., w_n)$ un vecteur de poids entiers associés aux n critères. La **norme de Tchebycheff** de s, aussi appelée max-regret de s, est donnée par :*

$$U(s) = max(\{w_i \times r_i(s) \mid i \in 1, ..., n\}).$$

*La solution s est préférée à la solution s' par la **méthode de Tchebycheff** si et seulement si $U(s) < U(s')$*

3 Calcul de solutions équilibrées

Le problème de décision associé à la gestion de dépendances étant NP-complet, il est possible d'utiliser un codage en logique propositionnelle de ce problème pour le résoudre [8, 1, 11, 18]. Dans un premier temps, on peut ainsi calculer la valeur optimale pour chaque critère, et obtenir $(c_1^*, c_2^*, ..., c_n^*)$. Le calcul d'une solution équilibrée se fait ensuite par ajout de contraintes au problème initial. Nous ne détaillons pas ici l'ensemble C de contraintes utilisées pour représenter les dépendances entre logiciels, elles sont décrites dans [1].

3.1 Résolution successive de problèmes de satisfaction

Le premier codage que nous avons employé consiste simplement à ajouter itérativement des contraintes au problème de départ pour écarter les solutions dont la norme de Tchebycheff est supérieure à un certain seuil, jusqu'à obtenir une solution dont la norme de Tchebycheff est minimale. On notera s^* une telle solution optimale pour la norme de Tchebycheff.

La valeur optimale $U(s^*)$ se caractérise facilement. Si r est une valeur telle que $C \cup \{w_i \times r_i(s) \leq r \mid i \in 1, ..., n\}$ est satisfiable, et que $C \cup \{w_i \times r_i(s) \leq r - 1 \mid i \in 1, ..., n\}$ ne l'est pas, alors on a $r = U(s^*)$.

Comme chaque critère que nous considérons dans le cadre de la gestion de dépendances de paquets Linux est une fonction linéaire de variables booléennes [1], les contraintes $w_i \times r_i(s) \leq r$ $(\forall i \in \{1, ..., n\})$ peuvent s'exprimer sous forme de contraintes pseudo-booléennes : $w_i \times c_i(s) \leq r + w_i \times c_i^*$ $(\forall i \in \{1, ..., n\})$.

Le principal avantage de cette méthode est d'être purement basée sur des contraintes en variables booléennes, ce qui permet d'utiliser un vaste choix de prouveurs pour résoudre le problème d'optimisation.

3.2 Transformation en un problème d'optimisation linéaire

La norme de Tchebycheff étant une fonction dans laquelle apparaît un maximum, elle a le désavantage de ne

pas être linéaire. Cela a pour inconvénient de ne pas nous permettre d'utiliser les outils d'optimisation de fonctions linéaires existants, et de devoir faire appel à des méthodes *ad hoc*, comme celle présentée ci-dessus.

Cependant, dans le contexte d'un problème codé sous forme de contraintes en variables entières, cet obstacle peut être surmonté sans trop de difficultés. Nous avons vu que l'on pouvait majorer la valeur de la norme de Tchebycheff des solutions du problème en majorant les regrets de chacun des critères considérés. Or, notre but est de minimiser la norme de Tchebycheff de nos solutions, et cette norme est exprimée simplement à l'aide d'une variable entière r. Nous obtenons donc le problème d'optimisation suivant :

$$\begin{array}{rl} \min & r \\ \text{tel que} & C \cup \{w_i \times r_i(s) \leq r \mid i \in 1, ..., n\}. \end{array}$$

Nous avions initialement écarté cette approche car elle nécessite l'utilisation d'une variable entière alors que nous avions décidé de travailler dans le formalisme pseudo-booléen. Cependant, il est possible de représenter r sous sa forme binaire [3], $r = \sum_{i=0}^{m-1} 2^i \times b_i$; on obtient alors le problème booléen suivant :

$$\begin{array}{rl} \min & \sum_{i=0}^{m-1} 2^i \times b_i \\ \text{tel que} & C \cup \{w_i \times r_i(s) \leq \sum_{j=0}^{m-1} 2^j \times b_j \mid i \in 1, ..., n\}. \end{array}$$

4 Algorithmes

4.1 Optimisation par renforcement de contraintes

Un premier algorithme d'optimisation itératif consiste à partir d'une solution quelconque, et à restreindre l'espace de recherche tant que c'est possible. C'est une approche classique pour les moteurs booléens d'optimisation [15]. On exploite ici le fait que la norme de Tchebycheff prend ses valeurs dans $\mathbb{N}$ pour nos problèmes. Ainsi, au départ, nous recherchons simplement une solution au problème, indépendamment des valeurs qu'elle donne aux différents critères. Soit r la valeur de son max-regret. On renforce alors les contraintes du problème en fixant $r - 1$ comme majorant pour chacun des $w_i \times r_i(s)$. Si une solution existe, alors on itère le processus en utilisant le max-regret de cette nouvelle solution comme nouvelle valeur pour r. Dans le cas contraire, on conclut qu'il n'existe pas de solution s telle que $U(s) \in [0, r-1]$, et par conséquent les solutions qui donnent un max-regret de r sont optimales.

Cet algorithme est correct dans notre cas car les valeurs prises par la fonction objectif admettent toutes un prédécesseur et un nombre fini de minorants.

Le temps de calcul de cet algorithme est fortement dépendant de la valeur de la solution trouvée à la première itération. En effet, si cette valeur est n, l'algorithme effectuera au maximum n itérations. Une amélioration possible consiste à rechercher une solution relativement proche de la valeur optimale dès le début. Pour cela, nous avons utilisé une fonction linéaire qui approche la valeur de notre fonction objectif, et nous avons optimisé selon cette fonction linéaire. Le temps de calcul d'une solution optimale pour U par ajout de contraintes peut être réduit significativement lorsque la solution initiale est une « bonne » solution pour notre fonction d'approximation. Cela permet également de donner un temps limite d'exécution au moteur d'optimisation mono-critère, afin d'éviter de perdre trop de temps lorsque celui-ci se rapproche de la valeur optimale pour l'approximation. L'intérêt de cette approche est de pouvoir conserver les clauses apprises entre deux appels successifs au prouveur car celles-ci sont conséquences logiques de l'ensemble de contraintes initial (SAT incrémental).

Entrées: problème P
Entrées: fonction objectif multi-critère U
Sorties: solution réalisable s minimisant U
$s \leftarrow obtenirSolution(P)$;
tant que $s \neq UNSAT$ **faire**
  $r \leftarrow maxRegret(U, s)$;
  $meilleure \leftarrow s$;
  $contraintes \leftarrow P \cup cstrMaxRegret(U, r)$;
  $s \leftarrow obtenirSolution(contraintes)$;
fin
retourner *meilleure* ;

Algorithm 1: Renforcement de contraintes

4.2 Optimisation linéaire binaire

La deuxième approche se place dans le contexte de l'optimisation d'une fonction objectif « binaire » de la forme $\min 2^0 b_0 + ... + 2^{m-1} b_{m-1}$. Pour ce genre de problème, un algorithme spécifique a été proposé [3], assurant un maximum de m appels au solveur SAT pour effectuer l'optimisation. Cet algorithme se base sur une optimisation dichotomique. En effet, la fonction objectif considérée peut prendre un maximum de 2^m valeurs. Afin de déterminer s'il existe une solution dont le coût est inférieur à 2^{m-1}, il suffit de chercher s'il existe une solution en fixant la variable b_{m-1} à la valeur 0. S'il existe une telle solution, alors on conserve cette valeur de vérité ; dans le cas contraire, on la fixe à la valeur 1. On effectue cette opération m fois, soit successivement pour les variables $b_{m-1}, ...b_1, ...b_0$.

Il est par ailleurs possible de sauter des étapes. Imaginons que le solveur trouve une solution de coût $c = \sum_{i=0}^{k} 2^i b_i + K$ pour un k entre $m-1$ et 0. Alors il est possible de conserver la valeur de vérité des b_i pour i de k à 0, tant que $b_i = 0$.

4.3 Optimisation par relaxation de contraintes

Il s'agit de l'approche duale de la première : on considère au départ une valeur de r qui minore $U(s^*)$. Si $C \cup \{w_i r_i(s) \leq r \mid i \in 1, ..., n\}$ possède une solution, alors cette solution est optimale. Dans le cas restant, on « relâche » les contraintes imposées en substituant $\{w_i r_i(s) \leq r \mid i \in 1, ..., n\}$ par $\{w_i r_i(s) \leq r + 1 \mid i \in 1, ..., n\}$ de manière à tester la valeur de r immédiatement supérieure à la précédente. On itère ce processus. Lorsqu'on trouve une solution, elle est nécessairement optimale (sinon on aurait trouvé une solution lors d'une itération passée). Cette approche rencontre un vif succès dans le cadre du problème d'optimisation MAXSAT lorsqu'elle est couplée à la détection de noyau incohérent [16]. Voir l'algorithme 2.

Entrées: problème P
Entrées: fonction objectif multi-critère U
Sorties: solution réalisable s minimisant U
$r \leftarrow 0$;
$contraintes \leftarrow P \cup cstrMaxRegret(U, r)$;
$s \leftarrow obtenirSolution(contraintes)$;
tant que $s = UNSAT$ **faire**
  $r \leftarrow r + 1$;
  $contraintes \leftarrow P \cup cstrMaxRegret(U, r)$;
  $s \leftarrow obtenirSolution(contraintes)$;
fin
retourner s ;

Algorithm 2: Relaxation de contraintes

Pour que cet algorithme soit correct, il faut disposer d'une valeur qui minore $U(s^*)$ (c'est le point de départ et dans notre cadre la valeur 0 convient), et faire varier r de façon à garantir que $U(s^*)$ sera atteint après un nombre fini d'itérations. Cela est le cas quand les fonctions de coût c_i sont à valeur dans $\mathbb{N}$ et que l'incrément choisi pour r est 1. L'intérêt de cette approche est de nécessiter exactement $U(s^*)$ appels au prouveur pour trouver une solution optimale : c'est intéressant pour les petites valeurs de $U(s^*)$. Contrairement aux cas précédents, il n'est pas possible de conserver les clauses apprises d'un appel à l'autre, ce qui exclut une approche collaborative par dichotomie.

5 Vers des solutions Pareto-optimales

Un des inconvénients majeurs dû à l'utilisation de la norme de Tchebycheff comme fonction d'agrégation de critères est que les solutions retournées ne sont pas nécessairement Pareto-optimales. En effet, considérons un problème bi-critère ayant deux solutions s_1 et s_2 optimales pour la norme de Tchebycheff. Supposons que ces solutions aient respectivement pour vecteurs de regrets $(2, 2)$ et $(2, 1)$. Les deux solutions ont effectivement un regret maximal de 2, mais on remarque que le deuxième vecteur domine le premier au sens de Pareto. Il est alors naturel de choisir la solution s_2. Or, aucun des algorithmes présentés plus haut ne peut assurer quelle sera la solution choisie. Nous proposons maintenant une méthode capable de pallier cet inconvénient.

L'obtention de solutions Pareto-optimales peut être garanti par un post-traitement, sous la forme d'une nouvelle fonction à optimiser. Cela fournit une méthode générique utilisable indépendamment de l'algorithme employé pour obtenir des solutions équilibrées. L'éventuel inconvénient pouvant survenir est une perte d'efficacité : le temps nécessaire au calcul de telles solutions serait peut-être moins important en modifiant directement nos algorithmes ou nos codages, et constitue une perspective de travail.

Afin de choisir une solution Pareto-optimale parmi les solutions Tchebycheff-optimales, on ajoute la contrainte suivante à notre problème (r est le max-regret minimal) :

$$C \cup \{w_i \times r_i(s) \leq r \mid i \in 1, ..., n\}.$$

Une fois ces contraintes ajoutées, il suffit de minimiser la fonction $\sum_{i=0}^{n} c_i(s)$ pour garantir la Pareto-optimalité.

Il est possible de combiner lexicographiquement la fonction linéaire présentée en section 3.2 et cette fonction d'optimisation assurant la Pareto-optimalité en une seule fonction linéaire. Cela permettrait d'obtenir un problème d'optimisation mono-critère. Cependant, cela se ferait au prix d'une perte de la prise en compte spécifique de la fonction linéaire « binaire ».

6 Expérimentations et résultats

Nous avons implémenté les différentes approches présentées dans une version modifiée de p2cudf [1], une adaptation du logiciel p2 [8] capable de résoudre des problème de gestion de dépendances pour GNU/Linux, basé sur Sat4j [7], une bibliothèque Java de prouveurs pour résoudre des problèmes de décision ou d'optimisation en variables booléennes. Nos expérimentations ont été réalisées sur un Quad-core Intel XEON X5550 avec 32Go de mémoire. Nous avons testé nos algorithmes sur des instances issues des compétitions *misc* [1] organisées dans le cadre du projet mancoosi. Nous nous sommes intéressés à trois séries d'instances : les *dudf-real* de la compétition *misc2011*, et les *9orless* et *10orplus* de la compétition *misc-live*. Il s'agit de problèmes réels d'installation de paquetages pour la distribution GNU/Linux Debian. Pour chaque instance, un temps limite d'exécution de dix minutes a été imposé.

Les tests ont été effectués sur deux (*removed*, *versionchanged*) puis trois (*removed*, *versionchanged*, *new*) critères à minimiser. *removed* correspond au nombre de pa-

1. http://www.mancoosi.org/misc/

quetages qui ont été enlevés du système, *versionchanged*[2] correspond au nombre de logiciels installés qui ont simplement changé de version et *new* correspond au nombre de logiciels non installés à installer.

	défaut	Pareto-opt.
renforcement	216	208
	15439s	18500s
linéaire binaire	206	203
	16085s	19381s
relaxation	214	206
	13220s	15593s
CPLEX	254	254
	21608s	22043s

FIGURE 1 – Récapitulatif des résultats des expérimentations (total de 254 instances, et temps de calcul cumulé de ces instances en secondes)

La figure 1 présente le nombre de problèmes résolus (sur 254) et le temps cumulé nécessaire pour résoudre ces problèmes. Nous considérons qu'un problème est résolu quand notre logiciel retourne une solution prouvée optimale par Sat4j. Il arrive très souvent que Sat4j trouve une solution optimale mais ne soit pas en mesure de prouver cette optimalité. C'est la raison pour laquelle l'outil p2cudf a obtenu de bons résultats lors de la compétition *misc2011*[3] car la preuve de l'optimalité n'est pas prise en compte. C'est un problème connu de Sat4j sur lequel nous travaillons. On peut noter tout d'abord que des trois codages que nous avons comparés, les deux basés sur l'ajout successif de contraintes donnent des résultats comparables. L'approche par linéarisation binaire est légèrement moins performante. En ce qui concerne la recherche de solutions Pareto-optimales, on remarque que le temps de calcul nécessaire à cette étape supplémentaire n'est pas négligeable. Avec le temps de calcul limité à dix minutes, les différentes versions de nos solveurs n'arrivent pas nécessairement à déterminer une solution Pareto-optimale alors qu'ils réussissent à calculer une solution Tchebycheff-optimale. Au niveau de la qualité des solutions, en revanche, on remarque une amélioration, surtout lorsqu'on considère les instances les plus difficiles (*10orplus*) avec trois critères. Voir la figure 2.

Nous avons aussi comparé nos approches en utilisant le solveur CPLEX 12 à la place de Sat4j. Pour ce solveur, nous avons utilisé le codage présenté en section 4.2 qui ne requiert pas de décomposition en puissances de deux la variable de la fonction objectif. En dépit du fait que nous communiquons avec CPLEX par ficihers et que cela a un coût non négligeable lors de l'échange d'informations avec

2. Le critère *changed* de Mancoosi correspond à *removed* + *versionchanged* + *new*. Nous avons introduit le critère *versionchanged* pour utiliser des critères indépendants.

3. `http://www.mancoosi.org/misc-2011/results/`

instance	défaut	Pareto	CPLEX pareto
epiphany-browser	75,23,18	75,23,17	75,23,17
	$751s$	$753s$	$132s$
evolution-dev	55,15,11	55,14,8	55,13,9
	$150s$	$224s$	$127s$
gnochm	39,4,5	39,3,4	38,4,4
	$101s$	$131s$	$121s$
gnome-panel-data	37,4,1	37,3,0	37,3,0
	$102s$	$136s$	$122s$

FIGURE 2 – Exemple de l'amélioration apportée par la recherche de solutions Pareto-optimales

p2cudf, CPLEX réussit à résoudre toutes nos instances, et rapidement pour la plupart d'entre elles. Ceci est cohérent avec les résultats obtenus par le prouveur UNSA lors de *misc2010.*

7 Conclusions et perspectives

Dans cet article, nous avons dans un premier temps comparé plusieurs codages et plusieurs algorithmes capables de résoudre des problèmes d'optimisation multi-critère sous contraintes pseudo-booléennes. Nous avons constaté que le codage pour lequel l'optimisation passe par plusieurs itérations « ajout de contraintes – test de cohérence » est plus efficace sur nos problèmes, quand on utilise notre solveur.

En revanche, nous n'avons pas pu départager les deux algorithmes que nous avons comparés. En effet, l'algorithme d'optimisation par renforcement de contraintes arrive à résoudre quelques instances de plus dans le temps imparti que celui qui optimise par relaxation de contraintes, mais il est généralement plus lent pour résoudre les instances que le solveur utilisant la relaxation lorsque ce dernier trouve une solution optimale.

Nous avons cherché à obtenir des solutions Pareto-optimales, ce qui n'est pas une propriété assurée par l'optimalité au sens de la norme de Tchebycheff. Pour cela, nous avons ajouté une étape de post-traitement qui permet d'obtenir une solution Tchebycheff-optimale Pareto-optimale. Cette étape se résume à une optimisation mono-critère : la minimisation de la somme des coûts associés à nos critères.

Nous avons aussi comparé nos approches en utilisant le solveur CPLEX à la place de Sat4j. CPLEX s'est révélé beaucoup plus performant. Cela ne remet pas en forcément en cause l'approche en variables booléennes. D'autres prouveurs en variables booléennes se rélèvent plus efficaces que Sat4j sur ces problèmes de gestion de dépendances (Clasp [5] et WBO [12] par exemple, voir les résultats de *misc2010* et *misc2011*). Nous utilisons Sat4j car nous maîtrisons complètement son fonctionnement, et parce que nous espérons aussi améliorer ses performances sur ce type de problèmes.

Une piste pour des travaux futurs est de comparer nos résultats avec d'autres méthodes d'optimisations multicritère, comme les OWR [14], qui ont l'avantage de fournir des solutions Pareto-optimales. Avec les OWR, il est possible de garantir la Tchebycheff-optimalité en plus de la Pareto-optimalité en considérant le poids associé au max-regret comme très important, voire même de se ramener à un leximax si les écarts entre les différents poids sont tous très grands.

Références

[1] Josep Argelich, Daniel Le Berre, Inês Lynce, João P. Marques Silva, and Pascal Rapicault. Solving linux upgradeability problems using boolean optimization. In Lynce and Treinen [9], pages 11–22.

[2] Daniel Le Berre, Emmanuel Lonca, Pierre Marquis, and Anne Parrain. Optimisation multicritère pour la gestion de dépendances logicielles : utilisation de la norme de tchebycheff. In *Actes de la conférence RFIA 2012*. RFIA, 2012.

[3] Michael Codish, Samir Genaim, and Peter J. Stuckey. A declarative encoding of telecommunications feature subscription in sat. In António Porto and Francisco Javier López-Fraguas, editors, *PPDP*, pages 255–266. ACM, 2009.

[4] R. Di Cosmo and S. Cousin. Project presentation. Technical report, Mancoosi (Managing the Complexity of the Open Source Infrastructure), 2008.

[5] Martin Gebser, Roland Kaminski, and Torsten Schaub. aspcud : A linux package configuration tool based on answer set programming. In Conrad Drescher, Inês Lynce, and Ralf Treinen, editors, *LoCoCo*, volume 65 of *EPTCS*, pages 12–25, 2011.

[6] G. Guttierez, M. Janota, I. Lynce, O. Lhomme, V. Manquinho, J. Marques-Silva, and C. Michel. Final version of the optimizations algorithms and tools. Technical report, Mancoosi (Managing the Complexity of the Open Source Infrastructure), 2011.

[7] D. Le Berre and A. Parrain. The sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7 :59–64, 2010.

[8] D. Le Berre and P. Rapicault. Dependency management for the eclipse ecosystem : eclipse p2, metadata and resolution. In *Proceedings of the 1st international workshop on Open component ecosystems*, pages 21–30. ACM, 2009.

[9] Inês Lynce and Ralf Treinen, editors. *Proceedings First International Workshop on Logics for Component Configuration*, volume 29 of *EPTCS*, 2010.

[10] F. Mancinelli, J. Boender, R. Di Cosmo, J. Vouillon, B. Durak, X. Leroy, and R. Treinen. Managing the complexity of large free and open source package-based software distributions. In *ASE*, pages 199–208. IEEE Computer Society, 2006.

[11] F. Mancinelli, J. Boender, R. Di Cosmo, J. Vouillon, et al. Managing the complexity of large free and open source package-based software distributions. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 199–208. IEEE Computer Society, 2006.

[12] Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Oliver Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009.

[13] Claude Michel and Michel Rueher. Handling software upgradeability problems with milp solvers. In Lynce and Treinen [9], pages 1–10.

[14] Wlodzimierz Ogryczak, Patrice Perny, and Paul Weng. On Minimizing Ordered Weighted Regrets in Multiobjective Markov Decision Processes. In Ronen I. Brafman, Fred S. Roberts, and Alexis Tsoukiàs, editors, *ADT*, volume 6992 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2011.

[15] Olivier Roussel and Vaso Manquinho. Pseudo-boolean and cardinality constraints. In *Handbook of Satisfiability*, pages 695–733. IOS Press, February 2009.

[16] João P. Marques Silva. Minimal unsatisfiability : Models, algorithms and applications (invited paper). In *ISMVL*, pages 9–14. IEEE Computer Society, 2010.

[17] R.E. Steuer and E.U. Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26(3) :326–344, 1983.

[18] Tommi Syrjänen. A rule-based formal model for software configuration. Research Report A55, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, December 1999.

[19] A.P. Wierzbicki. On the completeness and constructiveness of parametric characterizations to vector optimization problems. *OR Spectrum*, 8(2) :73–87, 1986.